Recall the definition of Maybe and Either:

```
data Maybe a = Nothing | Just a
```

```
data Either a b = Left a | Right b
```

1) Write a function that takes a list of positive and negative Float values. The function will apply the absolute value function to each value (abs). If the original value was positive then the resulting value should be an element of type Either Float Float of the form Left Float. If the original value was negative it should have the form Right Float. (Basically, we want to store all positive values, but store the information about the original value.)

```
e.g posNegStore [-1.0, -2.0, 3.4, 5.2, -3.2] =
    [Right 1.0, Right 2.0, Left 3.4, Left 5.2, Right 3.2]
```

2) The safeDiv function described a safe way to do division and return Nothing if the divisor is 0.

```
safeDiv x y
| y == 0 = Nothing
| otherwise = Just (x `div` y)
```

head, the function that takes the head a list, when the list is a cons of the form (x:xs). If head is applied to an empty list it returns an error.

```
head [] → error “not applicable”
```

Write a version of safeHead that returns Nothing if the list applied is empty or Just x if the list is non-empty.

3) Write a function that takes two functions f and g of type (a → c) and (b → d) respectively and applies them to an element of type Either a b. It will generate an element of type Either c d. Do it by analysis on the form of the Either argument (Left a or Right b).

```
e.g. applyEither (+1) (++)"stuff") (Left 5) = (Left 6)
```

4) Write a function that maps over a list of Either elements and applies the appyEither function to each element.

```
applyEitherMap:: (a → c) → (b → d) → [Either a b] → [Either c d]
```

Hand in a COMPILABLE Haskell file named hw6_<Your Name>.hs